Android 15 方案定制规范化
开发指南

版本号: 1.1
发布日期: 2024.11.15

版本历史

版本号	日期	制/修订人	内容描述
1.0	2023.03.21	AWA0989	初始版本文档
1.1	2024.11.15	AWA0989	针对 Android15 添加更新

目　录

插　图

1 概述

基于对标厂商 SDK 的分析，输出对 Allwinner SDK 模块配置、定制的规范性指导文档，使后续 Android 升级、模块开发、方案配置时更易于维护和修改。

⚠ 注意

文中出现禁止的条目，应当严格遵守，避免出现不可预测的结果。

2 当前现状

2.1 AW 方案配置主要变化

- **Android10 及之前**：与 Google 方案类似；
- **Android11 及之后**：分模块配置。使用 Makefile 的覆盖机制实现对模块的差异化修改。

2.1.1 SDK 代码组织结构

2.1.1.1 全志代码目录结构

Android11 开始，AW Android SDK 不再区分 BSP 与 Android 仓库集进行单独下载，只由一份 manifest 管理。Android12、Android13 及后续版本将与此保持一致。其目录结构上如下：

```
dailybuild@AwExdroid107:~/jenkins/code2/dailybuild/androidt/platform/android$ tree -L 1 .
.
├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
├── bootstrap.bash -> build/soong/bootstrap.bash
├── build
├── BUILD -> build/bazel/bazel.BUILD
├── cts
├── dalvik
├── developers
├── development
├── device
├── external
├── frameworks
├── hardware
├── kernel
├── libcore
├── libnativehelper
├── longan
├── out
├── packages
├── pdk
├── platform_testing
├── prebuilts
├── sdk
├── system
├── test
├── toolchain
├── tools
├── vendor
└── WORKSPACE -> build/bazel/bazel.WORKSPACE

27 directories, 4 files
```

图 2-1: SDK 代码结构

与 AOSP 相比，多了 longan 目录，用于存放 AW BSP 代码及编译打包工具，主要包括 spl、uboot、kernel、bsp 板级配置文件，以及编译打包工具。

从 Android14 起，是 AW 第一个采用 GRF 开发模式的 SDK。这种情况下，framework SDK 不再提供 longan 目录的代码，longan 相关代码由 vendor SDK 维护。

2.1.1.2 Android 配置与 BSP 配置的分界线

由于 Android11 及以上 Google 强制要求 kernel 使用 clang 进行编译，而编译 Android 专用 BSP 时，需要依赖 clang 等一些 AW BSP 目前不具备的工具。但这些工具在 AOSP 内通常是具有的，可以直接使用 AOSP 内的工具。因此，BSP 配置与 Android 配置必然存在一些耦合。目前：

- **BSP 专用的配置通常存放于**：longan/device/config/chips/<chip>
- **Android 专用的配置通常存放于**：device/softwinner/<product>

2.1.2 BSP 配置

BSP 配置通常包含如下项目：

- **BoardConfig-xx.mk**：编译配置，指定编译的内核版本、arch、defconfig、工具链等
- **xxx-defconf**：内核编译配置使用的 defconfig 文件
- **board.dts**：内核使用的 dts 文件
- **uboot-board.dts**：uboot 使用的 dts 文件
- **sys_config.fex**：主要存放 spl 使用的 dram 参数
- **sys_partition.fex**：bsp 平台专用分区配置文件
- **dragon_toc.cfg**：固件签名配置文件
- **env.cfg**：uboot env 配置文件

2.1.3 Android 配置

Android 的配置通常如下项目：

- **AndroidProducts.mk**：product lunch 配置
- **BoardConfig.mk**：基本的板级相关、硬件相关配置，如 product 分区配置，Wi-Fi、蓝牙硬件等
- **<product>.mk**：产品相关配置，包含要编译的 packages、属性、拷贝的资源文件等

- **overlay**：静态资源覆盖配置

除此外，其他文件或目录基本由 BoardConfig.mk、<product>.mk 来 include 或指定，以构成整个方案的完整配置。

2.1.4 由 Android 管理的 BSP 配置项目

由于 Android 专用 BSP 需要某些 Android 的资源，且由于 Android 每个版本均可能存在差异，bsp 单个基础开发分支无法完成兼容。因此，如下项目已经由 Android 来管理，不再由 BSP 板级配置管理：

- **dragon_toc.cfg**
- **sys_partition.fex**
- **env.cfg**
- **bsp-config.mk**：Android 专用 BSP 编译时的差异化配置

  如需要获取当前编译的 sdk 内由 Android 管理的配置，可以通过如下命令来确认：

```
get_build_var BOARD_ADD_PACK_CONFIG | xargs -n 1
```

3 方案配置规范化

3.1 整体原则

3.1.1 模块化配置

一个模块的配置尽量放在同一级目录下，使配置的层次结构清晰。如涉及与多个模块的交互无法完全独立，应当在模块配置说明文档描述清楚对应功能项所依赖的模块、配置项目等。

模块的配置，包括编译控制变量、属性设置、manifest 文件、initrc 及要编译的 packages 等。

3.1.2 配置公共部分

公共部分放在 common 目录下，新增板级配置时只需修改定制化的内容。为保证方案配置可以覆盖 common 的相关变量，应优先考虑?= 的赋值方式。

3.1.3 配置说明与版本信息

在每个模块下应该以 markdown/txt 文档的方式说明该模块所有用户可更改的配置，包含配置变量作用、配置方法，以及当前版本、使用的 SDK 版本等信息。

如对于 ceres 平台，camera 公共配置目录 device/softwinner/ceres/common/camera 内，除了 camera 必要的配置文件、资源文件外，还应增加 ReadMe.md 或 ReadMe.txt 用于解释模块当前的版本信息、如何使用、如何配置等。

对于很简单的模块，如仅仅涉及 1 行变量配置，可以在对应的配置 mk 文件内加以说明。

3.2 配置规范

3.2.1 方案名称

Android 方案名应能体现该 product 的基本特性。如：arm/arm64？是否为 go 设备？

规则：<product>_<TARGET_ARCH>[_go]。

其中，TARGET_ARCH 意指 Android 采用的 32 位 or64 位，非内核配置。

- **正例**：ceres_p25_arm_go、ceres_p25_arm64、ceres_p25_arm
- **反例**：ceres_b10

3.2.2 变量命名

对于 mk 内的配置变量，应**全部大写**。AW 私有的，非 Android 原生的变量，应以：CONFIG_AW_开头命名。如变量不需要方案配置，仅仅在某个模块作用范围内，由该变量推导其他变量用，则可以放宽限制。

对于 mk 内的属性变量，应**全部小写**。AW 私有的，在 selinux 允许的命名规范后，全部加.aw. 以表明是 AW 私有的属性。

正例：CONFIG_AW_ENABLE_GKI、ro.build.aw.camera.version

反例：CONFIG_LOW_RAM_DEVICE、ro.build.DroidBoost.version

3.2.3 文件权限

除可执行的 shell 脚本、bin 文件外，方案配置目录内一律不允许出现 755 权限的文件。

- **反例**

```
diff --git a/ceres-p25/camera/camera.cfg b/ceres-p25/camera/camera.cfg
new file mode 100755
index 0000000..4b67cdc
--- /dev/null
+++ b/ceres-p25/camera/camera.cfg
@@ -0,0 +1,194 @@
+;--------------------------------------------------------------------------------
+; 用于camera的配置
+;
+; 采用格式:
+; key = key_value
```

图 3-1: 文件权限

3.2.4 编码方式

配置文件应使用 UTF-8 的编码方式。如 git show 等命令内出现乱码，请确认是否使用了错误的编码方式。因历史遗留原因出现的文件内乱码问题，应在改动该文件，或是新方案配置时予以纠正。

- **反例**

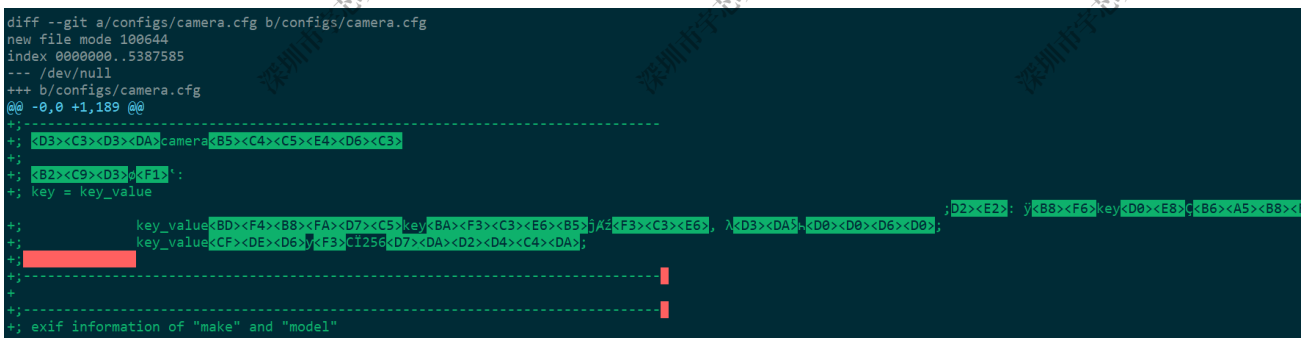

图 3-2: 文件编码

3.2.5 代码风格

1. **行尾空格**

   如非语法限制，所有文本类配置文件不应该以空格结尾。

- **反例**

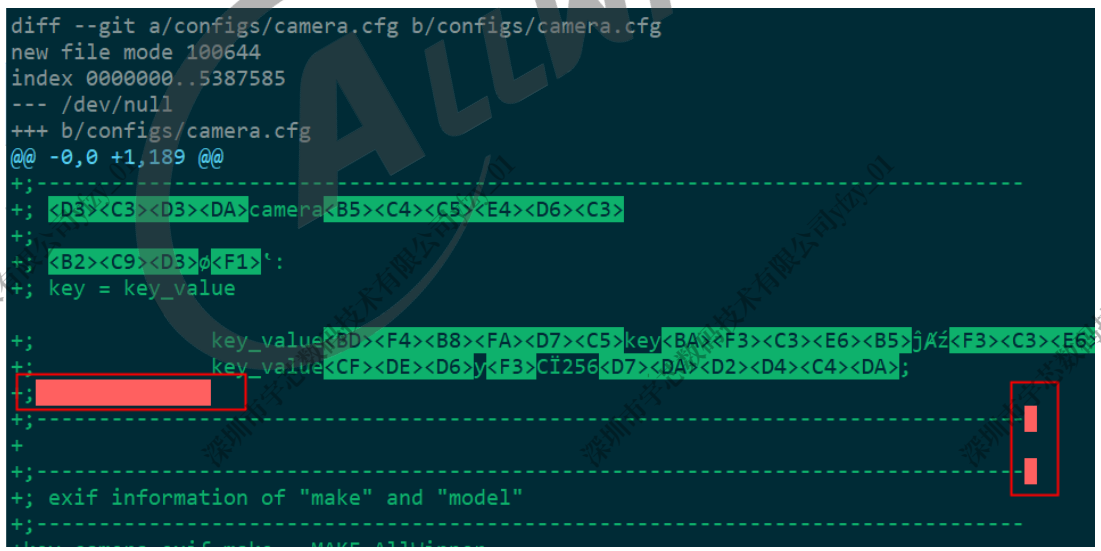

图 3-3: 行尾空格

2. **使用 Unix 风格的结尾符**

   文本类配置文件不应使用 Windows 风格结尾符（^M），而应当采用 Unix 风格的结尾符号（\n）。

- **反例**

```
diff --git a/overlay_go/packages/apps/DocumentsUI/res/values/dimens.xml b/overlay_go/packages/apps/DocumentsUI/res/values/dimens.xml
new file mode 100644
index 0000000..737e934
--- /dev/null
+++ b/overlay_go/packages/apps/DocumentsUI/res/values/dimens.xml
@@ -0,0 +1,3 @@
+<resources>^M
+    <dimen name="dialog_content_padding_top">0dp</dimen>^M
+</resources>^M
```

图 3-4: 结尾符

3. **缩进符**

方案配置内，Makefile 应统一使用空格作为缩进符，缩进大小：4 个字符。在不违反编码规范的前提下，新增代码应保持与附近代码一致的缩进规则。

- **反例**

```
57 else ifeq ($(filter-out img-%, $(GPU_ARCH)),)
58     PRODUCT_GPU_FILES += \
59         $(GPU_COPY_ROOT_DIR)/init.gpu.img.rc:$(TARGET_COPY_OUT_VENDOR)/etc/init/init.gpu.img.rc \
60 ›   $(GPU_COPY_ROOT_DIR)/init/android.hardware.graphics.allocator@4.0-service.img.rc:$(TARGET_COPY_OUT_VEN
61 ›   $(GPU_COPY_ROOT_DIR)/init/android.hardware.graphics.allocator@4.0.img.xml:$(TARGET_COPY_OUT_VENDOR)/et
62 ›   $(GPU_COPY_ROOT_DIR)/init/android.hardware.graphics.mapper@4.0-passthrough.img.xml:$(TARGET_COPY_OUT_V
63 ›   $(call find-copy-subdir-files,"powervr.ini",$(GPU_COPY_ROOT_DIR),$(TARGET_COPY_OUT_VENDOR)/etc) \
64         $(GPU_COPY_ROOT_DIR)/bin/pvrsrvctl:$(TARGET_COPY_OUT_VENDOR)/bin/pvrsrvctl \
product_config.mk
```

图 3-5: Makefile 缩进

4. **换行符**

如某个配置项需要使用换行符时，最后一行应禁止以换行符结尾，以避免其后没有空行时引发编译错误。

- **反例**

```
+
+PRODUCT_MAKEFILES := \
+    $(LOCAL_DIR)/ceres_b1.mk \
+    $(LOCAL_DIR)/ceres_b3.mk \
+    ceres_perf1:$(LOCAL_DIR)/ceres_perf1.mk \
+    ceres_perf2:$(LOCAL_DIR)/ceres_perf2.mk \
+    ceres_verify:$(LOCAL_DIR)/ceres_verify.mk \
+    ceres_qa:$(LOCAL_DIR)/ceres_qa.mk \
+
```

图 3-6: 换行符

改进方案：自动化编译检测确保，待部署。

3.2.6 禁止在源码目录下生成临时文件

涉及根据某些配置生成所需的配置文件时，应使用 local generated sources 机制将对应文件生成到 out 目录下的 local-generated-sources-dir，避免破坏源码结构。local generated sources 请参考 device/softwinner/common/misc 的编译。

- **反例**

device/softwinner/ceres/common/storage/config.mk

```
$(shell sed -i 's/,slotselect//g' $(TARGET_FSTAB))
```

3.2.7 禁止方案配置修改源码

禁止在方案配置内拷贝、修改某个 package 的 Android.mk、Android.bp 或对应的源文件。这种修改会引起源文件的变化，并引发编译逻辑的不确定性。

- **反例**

device/softwinner/mercury/common/media/config.mk

```
$(shell cp -rf
vendor/aw/public/framework/av/media/libaudioclient/aidl/android/media/
        AudioStreamType_ENABLED_AUDIO_STREAM_DSP.aidl
frameworks/av/media/libaudioclient/aidl/android/media/AudioStreamType.aidl)
```

3.2.8 禁止通过搜索 mk 的方式获取配置值

Makefile 内的变量值支持覆盖与条件判断，直接通过 grep 等方式搜索会导致结果不可靠。

- **反例**：hardware/aw/gpu/product_config.mk

```
TARGET_GPU_TYPE := $(shell grep TARGET_GPU_TYPE $(BOARD_CONFIG_FILE) | head -n 1 | cut -d = -f 2)
```

3.2.9 禁止通过修改编译逻辑增加全局 flag

编译 flag，如对编译平台 tablet、tv、auto 或厂商 allwinner、amlogic 的判断。

⚠ 注意

应修特别注意修改范围。全局 flag 的修改会引起所有代码重新编译，引起编译性能的降低。如有需求，请严格限制 flag 的作用范围。

- **反例**：build/make/core/binary.mk

```
# Add global definition for variable target platform

ifeq ($(TARGET_PLATFORM), tablet)

my_cflags += -DTARGET_PLATFORM_TABLET

else ifeq ($(TARGET_PLATFORM), homlet)

my_cflags += -DTARGET_PLATFORM_HOMLET

else ifeq ($(TARGET_PLATFORM), auto)

my_cflags += -DTARGET_PLATFORM_AUTO

endif
```

3.2.10 禁止有影响其他平台编译的配置

任何修改都不应该引起其他方案或平台编译 fail、功能异常。如某些模块的编译依赖具体平台，应该实现 default 逻辑的编译，即：没有定义该平台时，走 default 逻辑，避免编译出错。

- **正例**：hardware/aw/display/libdisplayd/include/platform.h

```
#else

#include "platform/platform-universal.h"
```

- **反例**：hardware/aw/display//hwc-hal/disp2/include/CompositionEngineV2Impl_defs.h

```
#elif defined(_board_petrel_)

#define CONFIG_MAX_DISPLAY_ENGINE 2

#define CONFIG_MAX_BANDWIDTH_LEVE 3

..

..

#else

+#error "Please add the corresponding platform definition"

+#endif
```

3.2.11 慎用 POST_INSTALL_CMD 编译选项

一个典型的例子，编译 ota image 时，POST_INSTALL_CMD 与生成 img 处于不同进程，POST_INSTALL_CMD 未完成时已经开始制作分区，这样可能会导致分区数据不正确。

- **反例**：frameworks/base/cmds/am/Android.mk

```
LOCAL_POST_INSTALL_CMD := mkdir -p $(TARGET_OUT_VENDOR)/framework; \
cp $(TARGET_OUT)/framework/am.jar $(TARGET_OUT_VENDOR)/framework/vendor_am.jar
```

3.2.12 逻辑判断

尽量减少 Makefile 内 if else 的判断的层数。当层数超过 3 时，代码将会变得逻辑难以理清。如无法避免较多的层级，确保正确的代码风格与注释，以帮助代码的阅读。

说明

逻辑判断层数较多时，建议重新梳理代码进行重构。

反例：hardware/aw/gpu/product_config.mk

```
117             $(GPU_COPY_ROOT_DIR)/lib64/gralloc.sunxi.so:$(TARGET_COPY_OUT_VENDOR)/lib64/hw/gralloc.$(TARGET_BOARD_PLATFORM
118         $(GPU_COPY_ROOT_DIR)/lib64/android.hardware.graphics.mapper@4.0-impl.so:$(TARGET_COPY_OUT_VENDOR)/lib64/hw/android
119     endif
120   endif
121 endif
122
product_config.mk
```

图 3-7: 逻辑判断

3.2.13 平台判断

默认方案配置内仅提供如下全局的属性/宏供各个模块使用：

表 3-1: 平台全局宏

变量名称	平板	盒子	车机	电视	投影仪	GSI	其他
PRODUCT_CHARACTERISTICS	tablet	homlet	auto	tv	optics	-	nosdcard
ro.build.characteristics	tablet	homlet	auto	tv	optics	default	nosdcard

默认提供芯片级全局的属性/宏供各个模块使用：

表 3-2: 平台全局属性

变量名称	A133	H616	T507	……
TARGET_BOARD_PLATFORM	ceres	cupid	mercuy	……
ro.board.platform	ceres	cupid	mercuy	……

对于更细一级的方案名称（如 ceres_p25_arm）类的变量，不应在模块内出现引用。

- **反例**：hardware/aw/display/hwc-hal/Android.mk

```
ifeq ($(TARGET_PLATFORM), auto)

LOCAL_SRC_FILES += disp2/GeneralDeviceFactory.cpp

else

ifeq ($(PRODUCT_DEVICE), ceres-t3)

ifeq ($(PRODUCT_DEVICE), apollo-p2)

LOCAL_SRC_FILES += disp2/GeneralDeviceFactory.cpp

else

LOCAL_SRC_FILES += disp2/TabletDeviceFactory.cpp
```

对于平台级、芯片级无关的配置，如模块版本、模块类型等板级相关的配置信息，可以放松限制。如：wpa_supplicant 版本：WPA_SUPPLICANT_VERSION。

⚠ 注意

变量命名应满足前述章节的规范。

3.2.14 Android.bp 编译控制

Android.bp 不像 Android.mk 一样可以很方便的获取方案内的编译配置。如果需要判断模块相关的一些配置信息，请使用.go 文件来实现编译流程的定制。关于 Android.bp 对于编译流程的控制，应遵循如下原则：

- **请勿修改 Android 原生的编译框架增加全局 flags** 的方式向模块传递变量。应当使用 soong_config_add 宏在方案目录内添加控制变量。使用方式 $(call soong_config_add, <namespace>,<key>,<value>))
- **为变量名设置合理的 namespace**，如隶属厂商平台的，应该命名为 allwinner；隶属于某个模块的，应当命名为其对应模块名称
- 模块编译控制 go 文件内使用 ctx.AConfig().VendorConfig(<namespace>).String(<key>) 的接口获取自定义的变量值

关于 go 文件对 bp 的编译控制使用方法，请参考文档《Build Control of Android.bp.docx》

3.2.15 模块编译信息提示

Android.mk、go 文件内应减少无关调试信息的打印，如当前编译控制变量的值、编译时所走的流程提示等。但如下情况必须添加打印以提示用户：

- 模块需要特殊的平台配置才能工作，当前模块源码里面不包含该配置，而使用了通用配置
- 有相同名称的资源文件，当前编译拷贝的文件无法确保是否正确，给出打印提醒用户
- 源码内缺少某些必要的资源文件、编译条件，但可以完成编译
- 已经弃用的变量、资源，在配置内仍被引用

⚠ 注意

打印应包含必要的信息，如存在问题的模块名称、设计的资源名称等。

- **正例**：Google 原生编译提醒：

build/make/core/soong_config.mk

```
build/make/core/soong_config.mk:209: warning: BOARD_PLAT_PUBLIC_SEPOLICY_DIR has been deprecated. Use SYSTEM_EXT_PUBLIC_SEPOLICY_DIRS instead.
build/make/core/soong_config.mk:210: warning: BOARD_PLAT_PRIVATE_SEPOLICY_DIR has been deprecated. Use SYSTEM_EXT_PRIVATE_SEPOLICY_DIRS instead.
```

图 3-8: 合理的编译警告

- **反例**：camera 模块编译提示信息：

hardware/aw/camera/hal_3/hal/allwinnertech/libAWIspApi/awTunningApp/Android.mk

```
hardware/aw/camera/hal_3/hal/allwinnertech/libAWIspApi/awTunningApp/Android.mk:3: warning: Use ceres awTuningApp!
```

图 3-9: 不必要的编译警告

3.2.16 模块运行调试等级控制

Android 已经实现了完整的机制来动态控制各个 LOG_TAG 的打印等级，模块无需再单独使用宏、属性来控制打印等级。使用方式：设置 persist.log.tag.<LOG_TAG> 为对应值以控制打印等级。目前支持的值及含义如下：

表 3-3: 原生打印等级控制

等级	值	英文	解释
0	S	silent	所有的都不打印
1	V	verbose	打印不低于 V 等级的 log
2	D	debug	打印不低于 D 等级的 log
3	I	info	打印不低于 I 等级的 log
4	W	warning	打印不低于 W 等级的 log
5	E	error	打印不低于 E 等级的 log

如：想要屏蔽 LOG_TAG 为 IMGSRV 的所有打印，请执行：

```
adb shell setprop persist.log.tag.IMGSRV S
```

反例：optee_client 模块打印等级控制：

hardware/aw/optee_client/tee-supplicant/Android.bp:

```
    cflags: [
        "-Werror",
        "-c",
        "-fPIC",
    ] + [
        "-DDEBUGLEVEL_1",
        "-DBINARY_PREFIX=\"TEES\"",
        "-DTEEC_LOAD_PATH=\"/vendor/lib\"",
        "-DTEE_FS_PARENT_PATH=\"/data/vendor\"",
    ] + ["-DCFG_TA_TEST_PATH=1"] + ["-DCFG_GP_SOCKETS=1"] + ["-DCFG_TA_GPROF_SUPPORT"],
```

图 3-10: 打印等级控制

3.2.17 模块自动识别

部分外设有自动识别的需求。对于该类外设，其开发规范应满足：

- 模块应该支持使用特定配置及自动识别两种配置，由编译阶段控制
- 使用自动识别配置时，其拷贝的静态资源文件在 sdk 内应当仅且只出现一次，避免多份文件带来额外的维护成本
- 自动识别编译控制文件对依赖的不同仓库下的厂商私有文件进行链接、拷贝、编译使用时，应检测其是否存在，避免不同 sdk 中不存在这些仓库时编译报错
- 烧写启动第一次完成识别，并保存配置。非第一次烧写启动时直接使用保存下来的配置，以避免启动时进行识别耗时
- 如果模块相关的功能跟系统 feature 相关，应实现机制，检测到模块无该 feature 时，禁用系统的 feature，以避免出现不必要的设置项、图标等。如：Wi-Fi/蓝牙模块的自动识别中，识别到不带蓝牙功能的模块时，应当禁用系统的 bluetooth feature，防止出现蓝牙相关的设置项，被客户错误操作

3.2.18 env 配置

env.cfg 最终将编译为系统的 env 分区。该分区决定了内核的启动参数，及驱动的配置参数。env.cfg 配置应该满足如下规范：

- 文件存在于 device/softwinner/<product>/common/system/env.cfg

- 在新 Android 版本开发时，应当仔细确认哪些参数是该版本已弃用不再使用了，哪些是新增的，力求保持文件的相对简介
- 同一平台的所有方案，如 a133，应当使用同一份配置，不区分版型。与版型相关的参数，应由其他机制保证（烧 key 等）

3.2.19 vendor_ramdisk 驱动

Android11 开始，为支持 GKI，Google 将大部分驱动 ko 化，要求将厂商相关的驱动编译为 ko 模块，打包到 vendor_boot 分区中的 vendor_ramdisk 中，由 init first stage 自动加载。为保证兼容性及维护的相当便利，vendor_ramdisk 中的模块应当满足如下规范：

- 由文件 vendor_ramdisk.modules 统一管理，编译阶段自动编译到 vendor_ramdisk 中
- 文件存在于 device/softwinner/<product>/common/system/vendor_ramdisk.modules
- vendor_ramdisk.modules 只存放 soc 相关，或者平台相关的驱动。soc 外的外设驱动，如 tp、Wi-Fi、sensor 等与板级相关的驱动，不应当存放于 vendor_ramdisk
- vendor_ramdisk.modules 对于同个平台的所有方案，如 a133，应当使用一份配置，不区分版型
- vendor_ramdisk.modules 支持以 # 为开头的注释，为维护方便，功能相同，或者相关的驱动应写到一起，并添加简短注释
- vendor_ramdisk.modules 中的 ko 在 first stage 加载时会自动寻找依赖关系，无需在该配置文件中手动排序（即使排序了也不生效）

⚠ 注意

SOC 无关的板级外设驱动，以及其他耗时较长的非必要驱动，应当放到/vendor/lib/modules，由用户自己控制及加载。

3.2.20 预编译文件管理

对于预编译文件的管理，统一放置在 vendor/aw/public/prebuild 目录下。如：

- 预编译动态库：vendor/aw/public/prebuild/lib
- 预编译 apk：vendor/aw/public/prebuild/apk
- 预编译二进制文件：vendor/aw/public/prebuild/bin

  对于预编译模块的存放，应满足如下规范：

1. 当在 AW private 存在对应预编译文件的源码时，且 AW private 仓库存在时，不应该引起模块名重复导致无法编译

2. 应当优先编译 private 仓库下的代码生成对应文件，以满足调试需求。具体可参考 libsoftwinner-ril 的实现
3. 提交预编译文件时，应在 git commit message 内说明该次更新主要改动点，及对应的源码仓库中的 git log 记录，如：

```
commit 26e306a721046ed39c33e9fd868072f0bfa7b127
Author: zhaoyouyi <zhaoyouyi@allwinnertech.com>
Date:   Fri Sep 30 10:05:05 2022 +0800

    fix: RIL: Vts test fail on android13.

    History Commit Message:

    0c19df3 feat: RIL: Vts support for android13 aidl implement.

    Scope: All
    IssueID: 101739

    Signed-off-by: zhaoyouyi <zhaoyouyi@allwinnertech.com>
    Change-Id: I9cbc8064a9580baffcf4b5d05c41d357c1239ef9
```

图 3-11: git 提交信息

3.2.21 属性变量及 Java 接口

Java 层使用属性作为模块开启/关闭的判断条件时，优先考虑 sysprop_library 模块，生成 Java API 供系统调用，而不是直接在 Java 代码内直接调用属性接口判断。

- **正例**：参考 system/libsysprop/src，关于系统属性与 api 的映射编译
- **反例**：frameworks/base/core/java/android/os/Build.java

```
+    /// AW CODE:[DroidBoost V2.0.6]droidboost control;jiangbin;200910
+    /**
+     * Whether use DroidBoost optimize.
+     * @hide
+     */
+    public static final boolean DROIDBOOST_ENABLED = !getString("persist.sys.droidboost.disable").equals("1");
+    /// AW:add end
+
```

图 3-12: 属性接口反例

3.2.22 动态资源覆盖

Android11 开始使用 mainline 模块。对于 mainline 模块内的资源文件，厂商使用 overlay 进行资源叠加覆盖时，必须使用动态 overlay，否则不生效。

AW 实现的 rro 模块，统一放置在 vendor/aw/public/package/rro 目录。以 TetheringConfig 模块为例说明其编写及使用规范：

- 按照 rro 编写规范，在 vendor/aw/public/package/rro/下实现所需模块的 rro apk，如 TetheringConfig；
- vendor/aw/public/package/rro/rro.mk 中加入该模块的编译：PRODUCT_PACKAGES += TetheringConfig；
- rro apk 内的默认资源资源配置应当与 mainline 模块内的值保持一致。即：vendor/aw/public/package/rro/TetheringConfig/res/values/config.xml 内所有值保持与其所覆盖模块中的资源值 packages/modules/Connectivity/Tethering/res/values/config.xml 一致；
- Product 方案配置实现对该 rro 模块的静态 overlay。对 ceres 方案，即：device/softwinner/ceres 下 common/overlay/overlay_rro/vendor/aw/public/package/rro/ TetheringConfig/res/values/config.xml 实现对 vendor/aw/public/package/rro/TetheringConfig/res/values/config.xml 的资源覆盖。

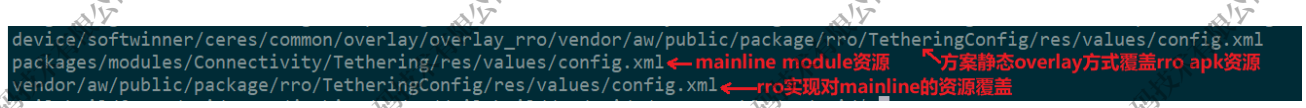

图 3-13: 动态 overlay

3.2.23 Bpf 覆盖

某些情况下，Android 发布的 bpf 文件可能存在异常。由于很多 bpf 程序打包在 mainline 模块内，使得厂商没有机会直接修改。为此，AW 实现了一套 bpf 覆盖机制，以实现对 mainline 模块 bpf 文件的修改。当然，对于非 mainline 模块的 bpf 程序，也可以使用该机制。

该机制的工作原理为：

- bpfloader 为 Google 原生用户空间 bpf 程序加载的服务，只加载特定目录下的 bpf 程序
- AW 对 bpfloader 修改，加载某个 bpf 程序时，判断特定的 bpf-override 目录是否也存在同名文件。如果存在，则使用 bpf-override 目录下的文件代替原有的 bpf 程序

```
--- a/bpfloader/BpfLoader.cpp
+++ b/bpfloader/BpfLoader.cpp
@@ -135,11 +135,20 @@ int loadAllElfObjects(const Location& location) {
         while ((ent = readdir(dir)) != NULL) {
             string s = ent->d_name;
             if (!EndsWith(s, ".o")) continue;

             string progPath(location.dir);
             progPath += s;

+            string systemOverridePath("/system/etc/bpf-override/" + s);
+            string vendorOverridePath("/vendor/etc/bpf-override/" + s);
+            if (access(vendorOverridePath.c_str(), F_OK) == 0) {
+                ALOGW("Override %s with %s", progPath.c_str(), vendorOverridePath.c_str());
+                progPath = vendorOverridePath;
+            } else if (access(systemOverridePath.c_str(), F_OK) == 0) {
+                ALOGW("Override %s with %s", progPath.c_str(), systemOverridePath.c_str());
+                progPath = systemOverridePath;
+            }
```

图 3-14: 加载自定义 BPF

AW 实现的 rro 模块，统一放置在 vendor/aw/public/bpf 目录。以 offload 模块为例说明其编写及使用规范：

- 按照 Android bpf 编写规范，重写或对原有 bpf 程序源码进行修改，并编译出新的 bpf.o 文件，放置在 vendor/aw/public/bpf/offload 目录下
- vendor/aw/public/bpf/offload 目录下应包含对该 bpf 程序的说明文件，如：程序来源、解决什么问题、如何编译等

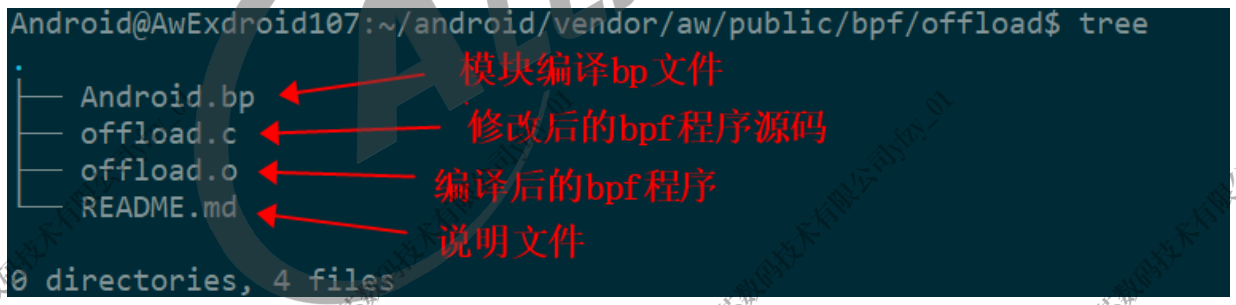

图 3-15: BPF 示例

- vendor/aw/public/bpf/bpf.mk 内加入对该模块的编译：PRODUCT_PACKAGES += prog-offload

3.2.24 编译阶段的 hook 机制

当厂商对编译有特殊的控制需求时，如在 Android lunch、make 等前后添加一些特殊的定制化动作，可以通过 AW 提供的 _hook_function_inject 机制对这些过程进行控制。详细实现方式请参考 device/softwinner/common/vendorsetup.sh

使用方法：

- 编写 hook 函数。通常，建议 hook 功能函数以 _ 开头，如 _make_start_hook、_make_end_hook 等
- 添加对原有功能的 hook：

```
_hook_function_inject -f &lt;func to hook&gt; -s &lt;func start hook&gt; -e &lt;func end hook&gt; -a &lt;action&gt;
```

特别的，AW 实现了对 make 的 hook，以方便用户在方案内直接使用。其控制变量为：

CONFIG_AW_PREBUILD_SCRIPTS：在 make 之前要执行的动作

CONFIG_AW_POSTBUILD_SCRIPTS：在 make 之后要执行的动作

⚠ 注意

默认的 hook 有对返回值的判断，CONFIG_AW_PREBUILD_SCRIPTS 或 CONFIG_AW_POSTBUILD_SCRIPTS 执行失败时将返回错误码，并终止编译。

3.2.25 GKI Boot Image 配置

Android13 起，GMS 方案 google 强制要求使用 gki boot image 镜像生产，否则过不了 GMS 测试。对于 GKI boot image，使用 CONFIG_AW_ENABLE_GKI 变量来控制。如需使用自己编译的 boot image 进行调试，以 A133 (ceres) 为例，需要按如需方式修改：

- 更改代码：将 device/softwinner/ceres/BoardConfig.mk 中 CONFIG_AW_ENABLE_GKI 改为 false：

```
CONFIG_AW_ENABLE_GKI := false
```

作用范围：永久有效

- 设置当前 shell 的环境变量：

```
export CONFIG_AW_ENABLE_GKI=false
```

作用范围：仅对当前 shell 的编译有效

- 编译期间传递参数：

```
CONFIG_AW_ENABLE_GKI=false make -j16
# 或
make -j16 CONFIG_AW_ENABLE_GKI=false
```

作用范围：仅对该次编译有效

特别的，当 device/softwinner/ceres/BoardConfig.mk 内指定的 GKI 镜像（BOARD_PREBUILT_BOOTIMAGE 变量）找不到时，即使配置了 CONFIG_AW_ENABLE_GKI := true，也会采用 aw 自己编译的 boot image 镜像，并且编译时打印如需信息：

```
OUT_DIR=out
============================================
16:21:40 Disallowed PATH tool "make" used: []string{"make", "events_xml.h", "defaults_xml.h", "SrcMd5.cpp"}
16:21:40 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
wildcard(vendor/partner_modules/gki/boot-5.15.img) was changed, regenerating...
[100% 1/1] initializing build system ...
device/softwinner/ceres/BoardConfig.mk:34: warning: GKI image vendor/partner_modules/gki/boot-5.15.img not found, disable GKI boot image
[  0% 2/295] including out/soong/Android-ceres_p25_arm_go.mk ...
    0:17 including out/soong/Android-ceres_p25_arm_go.mk ...
```

图 3-16: GKI 镜像找不到警告打印

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。